System design: YouTube

Suresh Podeti
6 min read · Sep 30, 2023

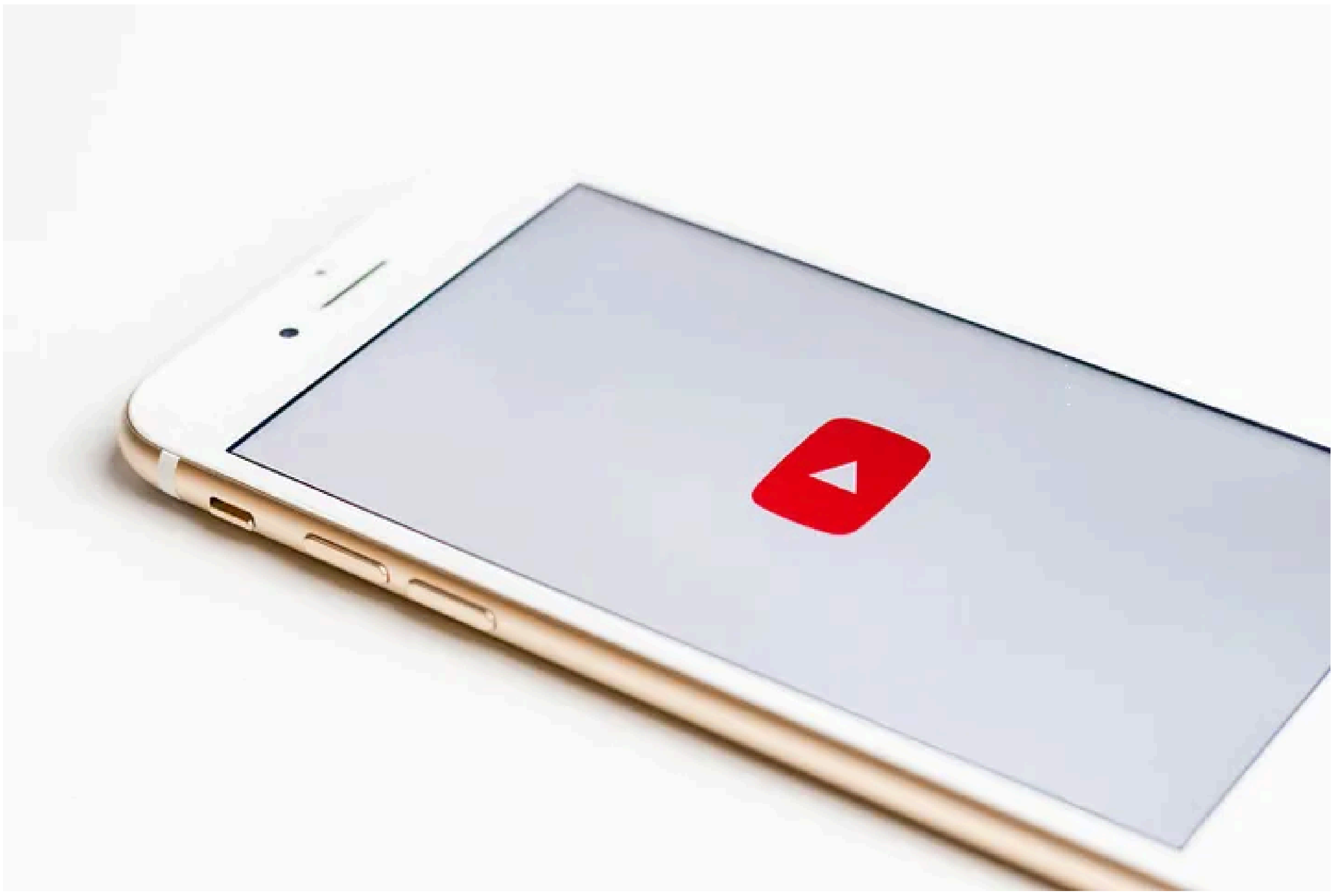


Photo by Sara Kurfeß on Unsplash

Introduction

YouTube is a popular **video streaming service** where users upload, stream, search, comment, share, and like or dislike videos. Large businesses as well as individuals maintain channels where they host their videos. YouTube allows free hosting of video content to be shared with users globally. YouTube is considered a primary source of entertainment, especially among young people.

Requirements

Functional

We require that our system is able to perform the following functions:

1. Upload videos (with video file, thumb nail, and title)
2. Search videos according to titles
3. Stream videos
4. Like and dislike videos
5. Add comments to videos
6. View thumbnails

Non-functional

- **High availability**: The system should be highly available. High availability requires a good percentage of uptime. Generally, an uptime of 99% and above is considered good.
- **Scalability**: As the number of users grows, these issues should not become bottlenecks: storage for uploading content, the bandwidth required for simultaneous viewing, and the number of concurrent user requests should not overwhelm our application/web server.
- **Good performance**: A smooth streaming experience leads to better performance overall.
- **Reliability**: Content uploaded to the system should not be lost or damaged.
- **Consistency**: We don't require strong consistency for YouTube's design. Consider an example where a creator uploads a video. Not all users subscribed to the creator's channel should immediately get the notification for uploaded content.

High level design

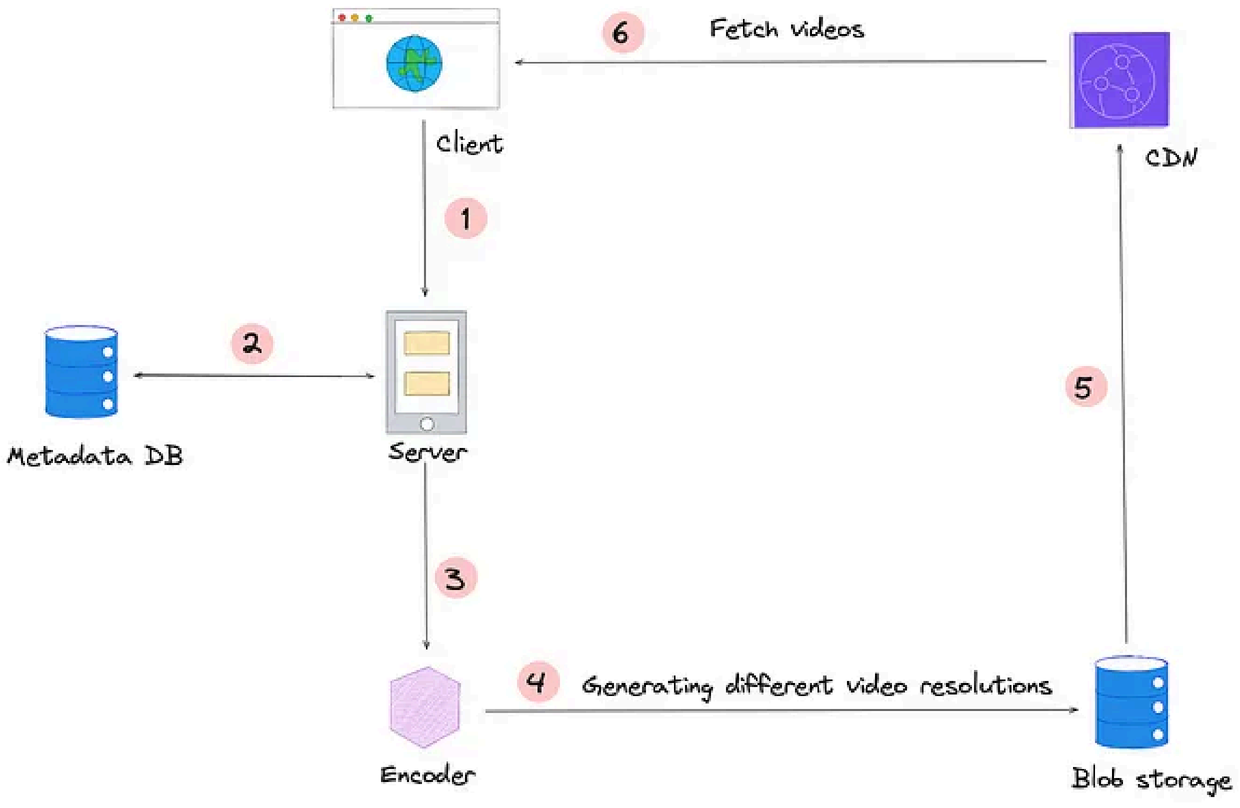


Fig 1.0: High level design of YouTube

API Design

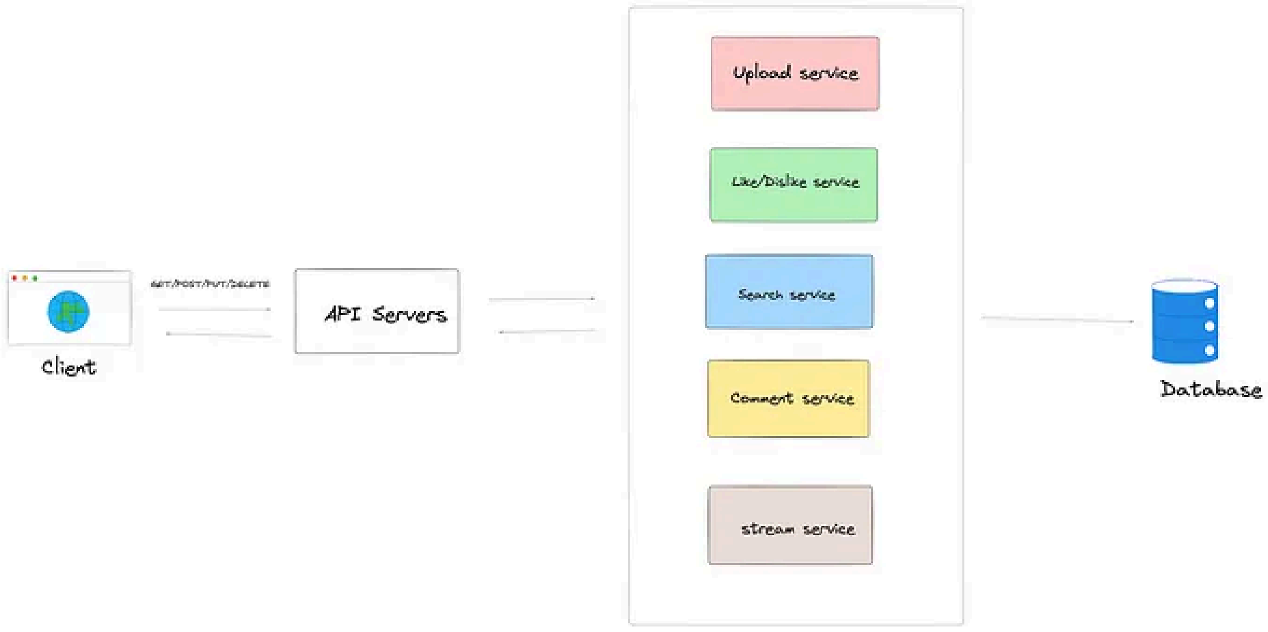
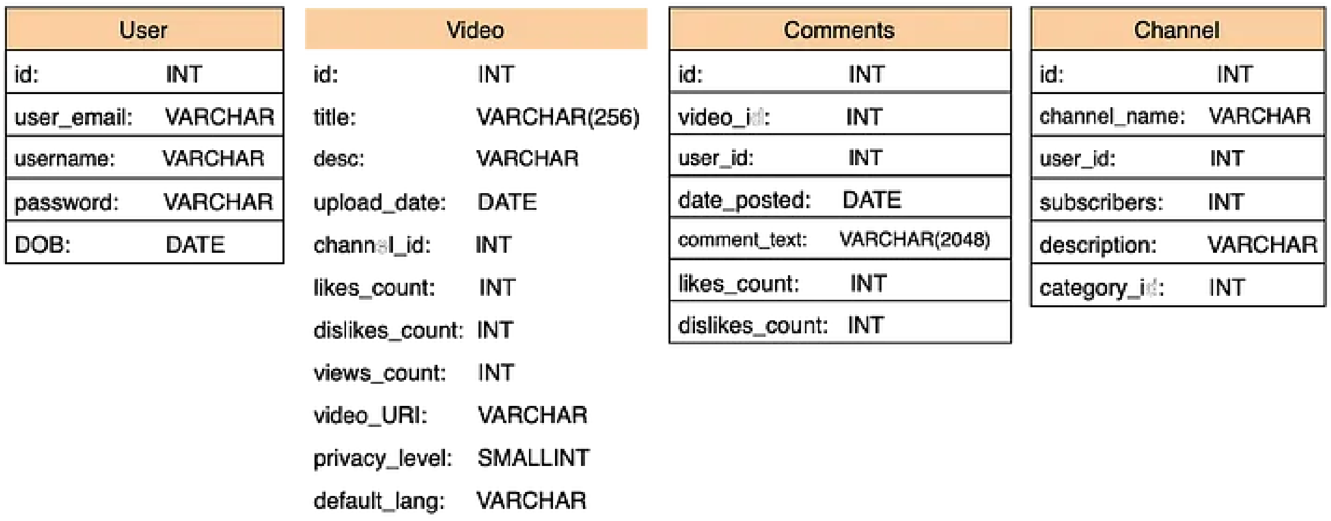


Fig 2.0: Different services in youtube design

Database schema



User	
id:	INT
user_email:	VARCHAR
username:	VARCHAR
password:	VARCHAR
DOB:	DATE

Video	
id:	INT
title:	VARCHAR(256)
desc:	VARCHAR
upload_date:	DATE
channel_id:	INT
likes_count:	INT
dislikes_count:	INT
views_count:	INT
video_URI:	VARCHAR
privacy_level:	SMALLINT
default_lang:	VARCHAR

Comments	
id:	INT
video_id:	INT
user_id:	INT
date_posted:	DATE
comment_text:	VARCHAR(2048)
likes_count:	INT
dislikes_count:	INT

Channel	
id:	INT
channel_name:	VARCHAR
user_id:	INT
subscribers:	INT
description:	VARCHAR
category_id:	INT

Fig 3.0: Database schema

Detailed design

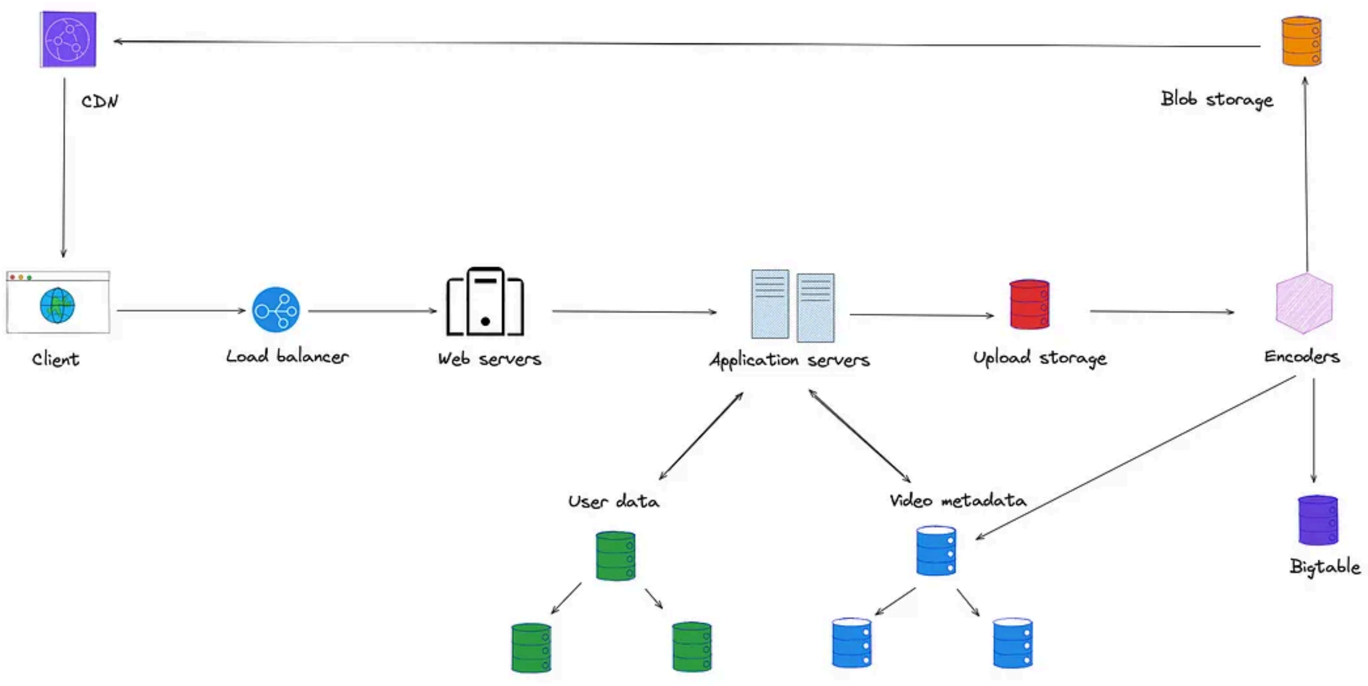


Fig 4.0: Detailed design of YouTube's components

Multiple storage units are used. Let's go through each of these:

- **Upload storage** is temporary storage that can store user-uploaded videos.
- **User account data** is stored in a separate database, whereas **videos metadata** is stored separately. The idea is to **separate the more frequently and less frequently accessed storage** clusters from each other for optimal access time.

- **Bigtable** — For each video, we'll require multiple thumbnails. Bigtable is a good choice for storing **thumbnails** because of its high throughput and scalability for storing key-value data. Bigtable is optimal for storing a large number of data items each below 10 MB. Therefore, it is the ideal choice for YouTube's thumbnails.
- **Encoders** — The encoders generate thumbnails and also store additional metadata related to videos in the metadata database. It will also provide popular and moderately popular content to CDNs and colocation servers, respectively.

YouTube search

Each new video uploaded to YouTube will be processed for data extraction. We can use a JSON file to store extracted data, which includes the following:

- Title of the video.
- Channel name.
- Description of the video.
- The content of the video, possibly extracted from the transcripts.
- Video length.
- Categories.

Each of the JSON files can be referred to as a document. These documents are indexed and used for searching a video quickly. We have already discussed how the distributed search works in following article.

Distributed Search
Introduction
medium.com

The approach above is simplistic, and the relevance of keywords is not the only factor affecting search in YouTube. In reality, a number of other factors will matter. The processing engine will improve the search results by filtering and ranking videos. It will make use of other factors like view count, the watch time of videos, and the context, along with the history of the user, to improve search results.

Encode

The raw videos uploaded to YouTube have significant storage requirements. It's possible to use various encoding schemes to reduce the size of these raw video files, allowing for efficient streaming over the internet. Smaller file sizes result in faster loading times and reduced buffering.

Since multiple devices could be used to stream the same video, we may have to encode the same video using different encoding schemes resulting in one raw video file being converted into multiple files each encoded differently.

We'll divide the **video into smaller time frames** and **encode them individually**. We can divide videos into shorter time frames and refer to them as **segments**. **Each segment** will be **encoded** using **multiple encoding schemes** to generate different files called **chunks**. The choice of encoding scheme for a segment will be based on the detail within the segment to get **optimized quality with lesser storage requirements**.

Encoding a video frame by frame, or using a technique known as **chunked encoding.** It enables adaptive streaming, allowing the platform to encode video segments at various bitrates and resolutions for a seamless user experience.

Using the strategy above, we'll have to encode individual shots of a video in various formats. However, the alternative to this would be storing an entire video (using no segmenting) after encoding it in various formats. If we encode on a per-shot basis, we would be able to optimally reduce the size of the entire video by doing the encoding on a granular level.

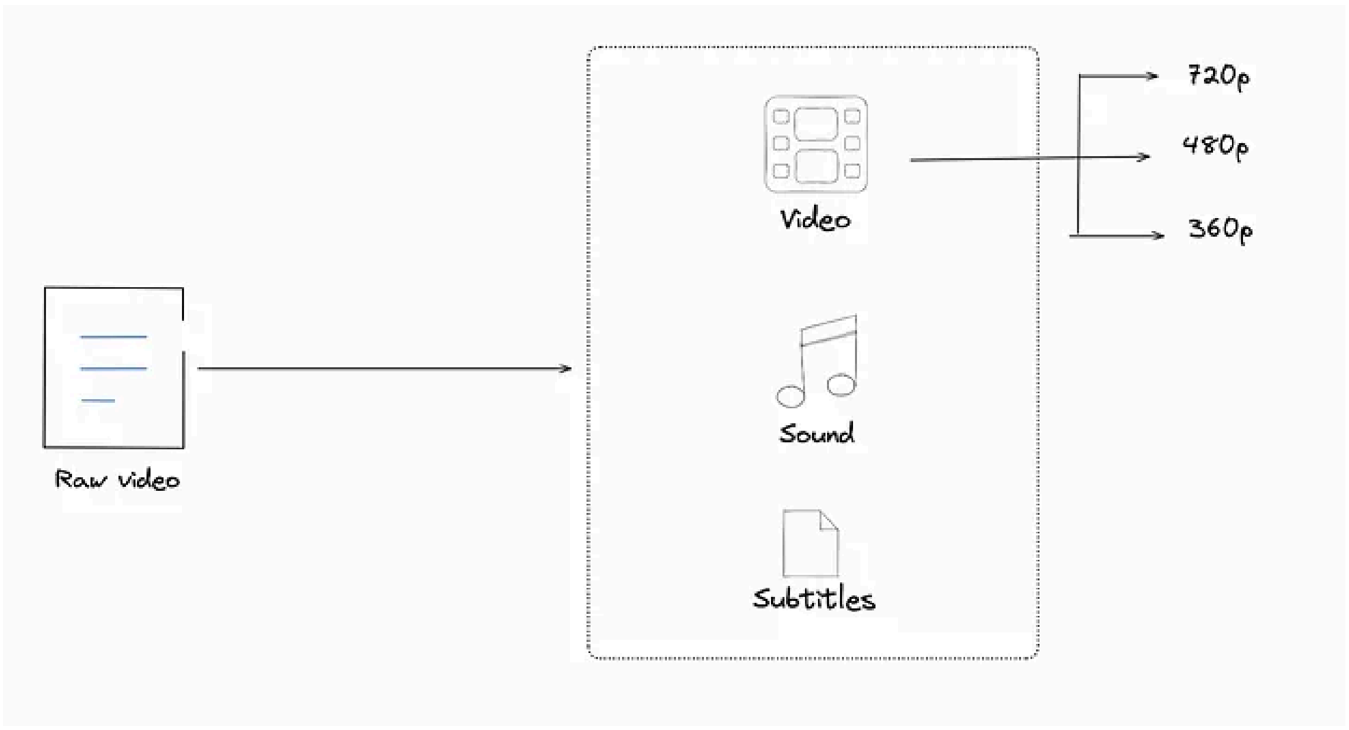


Fig 5.0: A raw video file being encoded into different formats including its audio and subtitles

Deploy

We have to bring the content closer to the user. This has three main advantages:

1. Users will be able to stream videos quickly.
2. There will be a reduced burden on the origin servers.
3. Internet service providers (ISPs) will have spare bandwidth.

So, instead of streaming from our data centers directly, we can deploy chunks of popular videos in CDNs and point of presence (PoPs) of ISPs.

Our content can be placed in internet exchange point (IXPs). We can put content in IXPs that will not only be closer to users, but can also be helpful in filling the cache of ISP PoPs.

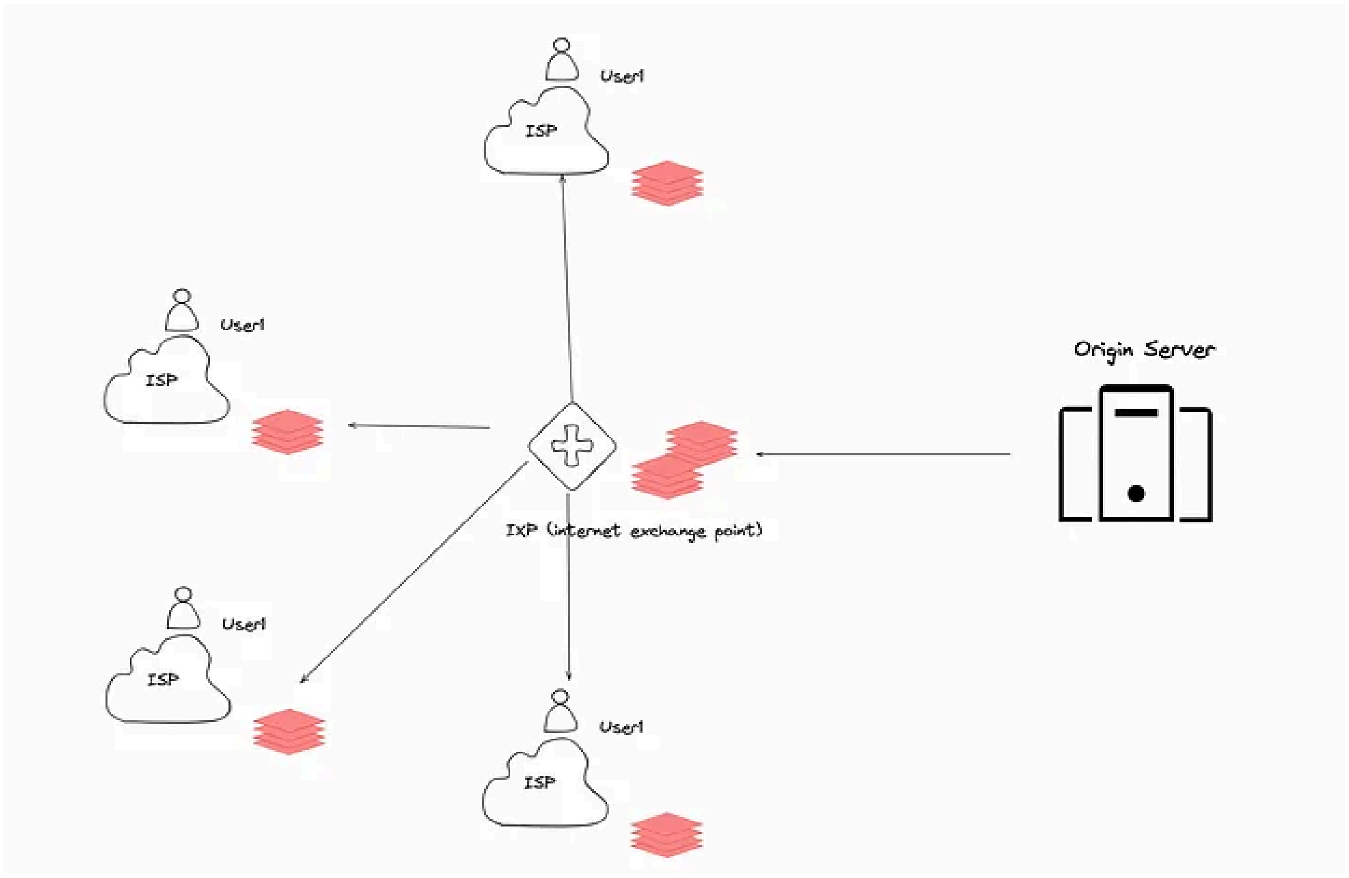


Fig 6.0: Streaming from data center to users through IXP and ISPs

We should keep in mind that the caching at the ISP or IXP is performed only for the popular content or moderately popular content because of limited storage capacity. Since our per-shot encoding scheme saves storage space, we'll be able to serve out more content using the cache infrastructure closer to end users.

Additionally, we can have two types of storage at the origin servers:

1. **Flash servers:** These servers hold popular and moderately popular content. They are optimized for low-latency delivery.
2. **Storage servers:** This type holds a large portion of videos that are not popular. These servers are optimized to hold large storage.

Deliver

Let's see how the end user gets the content on their device. Since we have the chunks of videos already deployed near users, we redirect users to the nearest available chunks.

Adaptive streaming

While the content is being served, the bandwidth of the user is also being monitored at all times. Since the video is divided into chunks of different qualities, each of the same time frame, the chunks are provided to clients based on changing network conditions.

when the bandwidth is high, a higher quality chunk is sent to the client and vice versa

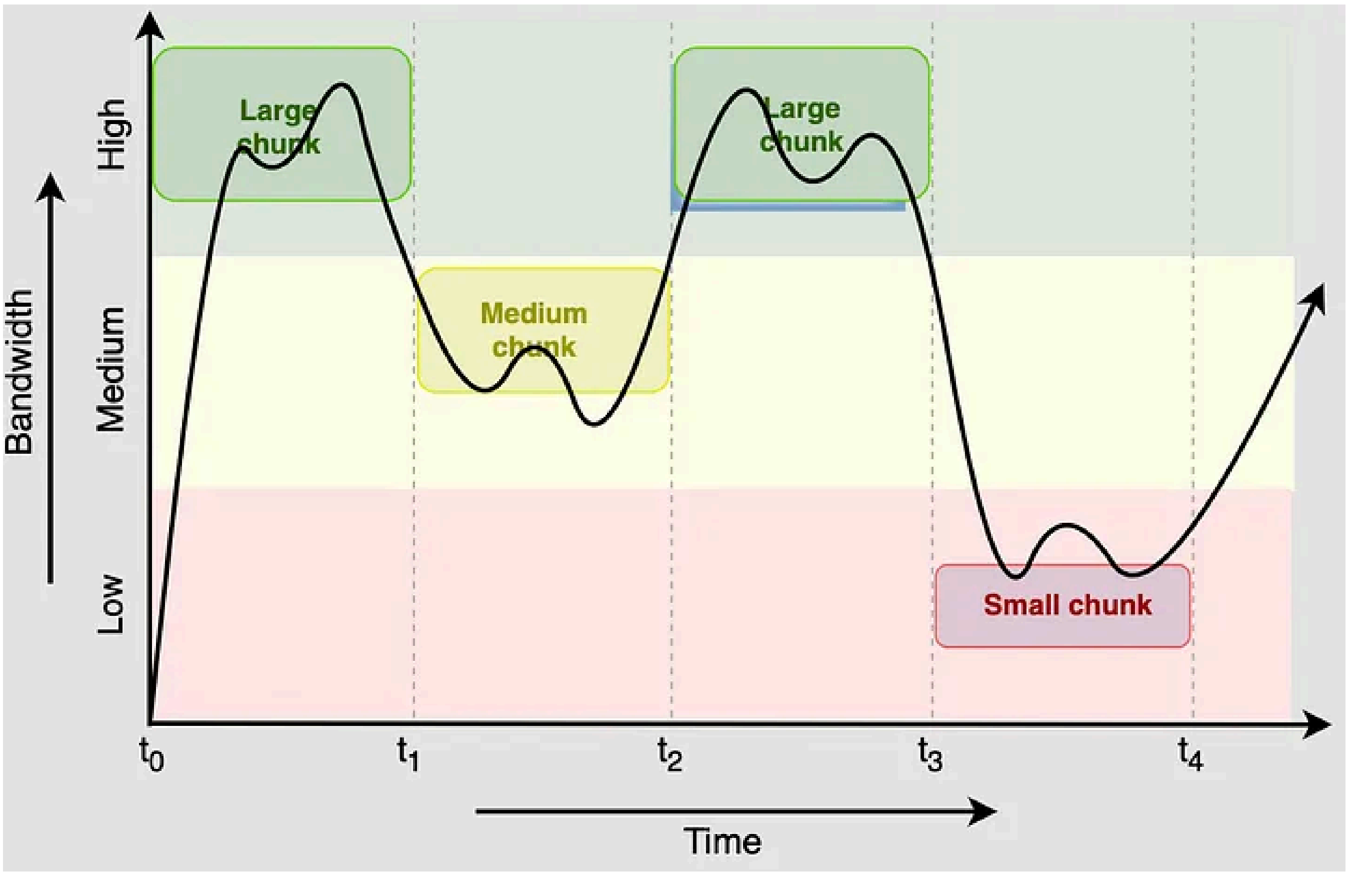


Fig 7.0: Chunk size in each time frame provided to the client changes according to the bandwidth

YouTube Software Architect Software Architecture Distributed Systems

System Design Interview

Edit profile